Grundlagenpraktikum Rechnerarchitektur

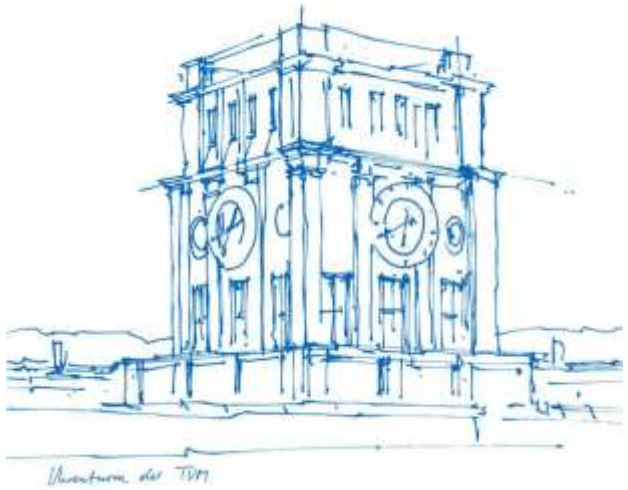
Team 230 – Vortrag zu Aufgabe A204

„Bilddekompression“

Sommersemester 2022

Eslam Nasrallah **Vladislav Dzhorov** **Yiyang Xie**

Lehrstuhl für Rechnerarchitektur und Parallele Systeme
Fakultät für Informatik
Technische Universität München



Inhalt

1. Einleitung
2. Lösungsansatz
3. Korrektheit
4. Performanzanalyse
5. Zusammenfassung

Einleitung

- **Datenkompression** - Codierung von Informationen mit weniger Bits als die ursprüngliche Darstellung, kann verlustfrei oder verlustbehaftet sein

- **Bildkompression – Grundprinzip** – statt jedes Pixel mit Farbe F = {R, G, B} einzeln zu speichern, Folgen von gleichen Pixeln mit Tupeln der Form (N, F) repräsentieren – N mal Pixel mit Farbe F

Ergebnis

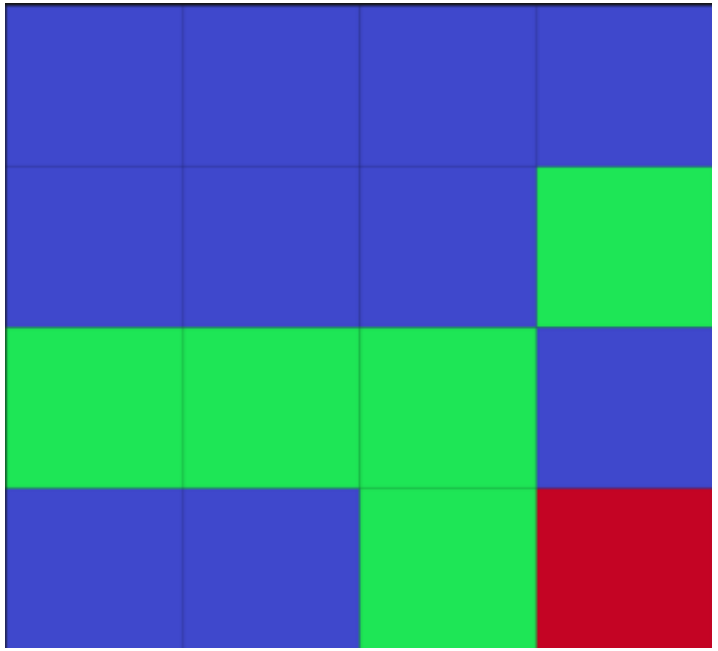
Annahme:

N und F kosten jeweils 1 Speichereinheit:

komprimiert = 7B4G3B1G1R

unkomprimiert = BBBBBBBGGGGBBBGR

=> 7 Speichereinheiten gespart



Einleitung - Fortsetzung

- **Aufgabenstellung** – schon komprimierte Bilder zu dekomprimieren
- **Bilderformat** – Das BMP Datei Format, auch als Bitmap bekannt, 8 Bits pro Pixel
- **Komprimierungsverfahren** – 8 bit Run-Length Encoding (RLE), eine verlustfreie Datenkompressionsmethode. RLE operiert in 2 Modi – “encoded mode” und “absolute mode”, hat auch 3 “escape characters”,

[03 04] [05 06] [00 03 45 56 67 00] [02 78] [00 02 05 01] [02 78] [00 00] [09 1E] [00 01]

---

04 04 04

06 06 06 06 06

45 56 67

78 78

Aktualisiere aktuelle position um 5 nach rechts und 1 nach oben

78 78

Zeilenende

1E 1E 1E 1E 1E 1E 1E 1E 1E

Bitmap endet

Lösungsansatz - Rahmenprogramm

- **Struktur von BMP Dateien –** relevant sind Header, InfoHeader und PixelData

Structure	Corresponding Bytes
Header	0x00 - 0x0D
InfoHeader	0x0E - 0x35
ColorTable	0x36 - variable
Pixel Data	variable

  - **Header –** DataOffset
  - **InfoHeader –** Width, Height, Bits per Pixel, Compression, ImageSize
  - **Pixel Data –** komprimierte Pixeldaten

Lösungsansatz - Fortsetzung

- **Funktionsheader:**

void bmp_rld(const uint8_t *rle_data**, size_t** len**, size_t** width**, size_t** height**, uint8_t** *img**)**

- **Algorithmus - Pseudocode**

```
WHILE inputFileLength > 0
        IF firstByte == 0
                SWITCH secondByte
                        case 0: linebreak
                        case 1: end of bitmap
                        case 2: delta
                        default: absolute mode -> copy secondByte pixels
                ENDSWITCH
        ELSE
                encoded mode -> copy secondByte firstByte times
        ENDIF
ENDWHILE
```

Lösungsansatz - Implementierungen

- **Naiver Ansatz – C -** jedes Byte einzeln mit einer for-Schleife kopieren
- **Naiver Ansatz – x86-64 Assembler –** wie C, aber im Assembler
- **Assembler-Implementierung, optimiert mit SIMD –** 16 Bytes zusammen durch SIMD Instruktionen kopieren
- **Implementierung in C mit memset()** – alles, was kopiert werden muss, auf einmal kopieren

Lösungsansatz - Optimierungen

- **Erste Optimierungsidee**

```
while (rep >= 16) {
        vx = _mm_loadu_si128(&num);
        vx = _mm_shuffle_epi8(vx, mask);
        _mm_storeu_si128(&img[j], vx);
}
```

- **Weitere Versuche**

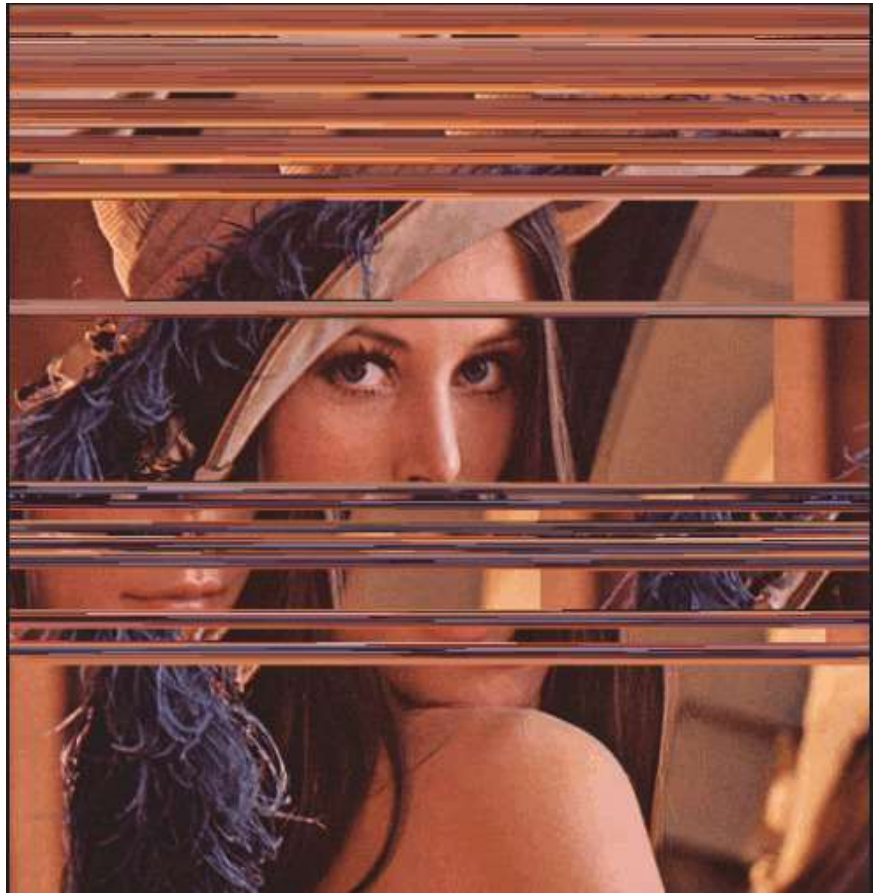
- **Verwendung von memset und memcpy**

```
{
        rep = rle_data[i++];
        num = rle_data[i++];
        linecounter += rep;
        memset(&img[j],num,rep);
        j+=rep;
}
```

Korrektheit

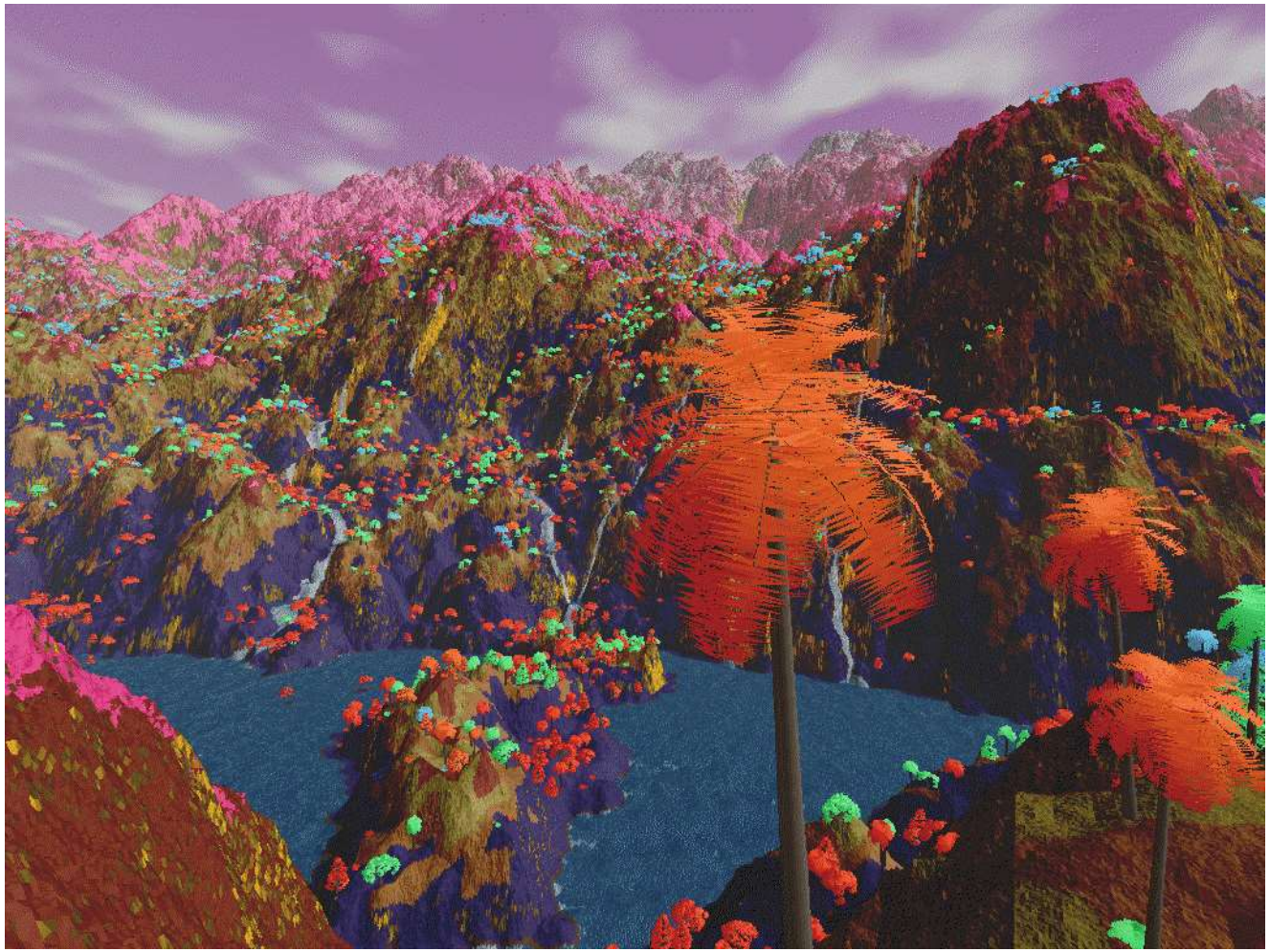
- **Testcases erstellen –** testcases mit Photoshop generieren
- **Fehler Beispiele –** nächste Folie
- **Behandlung spezieller Marker** – nicht farbbezogene oder Standardfarbe-Bytes mit 0 Byte laden; Delta und Linebreak behandeln
- **Behandlung von Byte Allignment** – 4-Byte Alignment für jede Zeile
- **Konkrete Behandlungen** – **uint8_t** pass**, uint8_t** linecounter**, uint8_t** linePadd

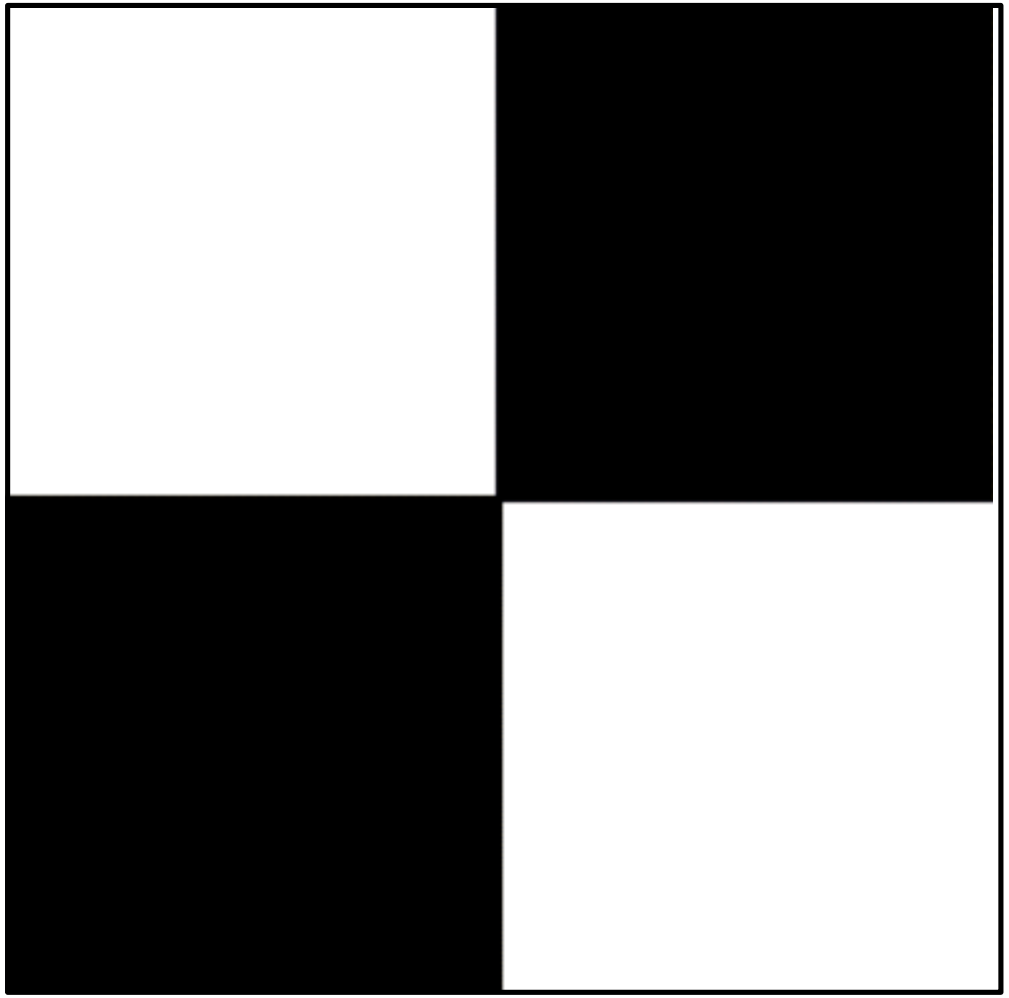
Korrektheit – Fehlerbeispiele



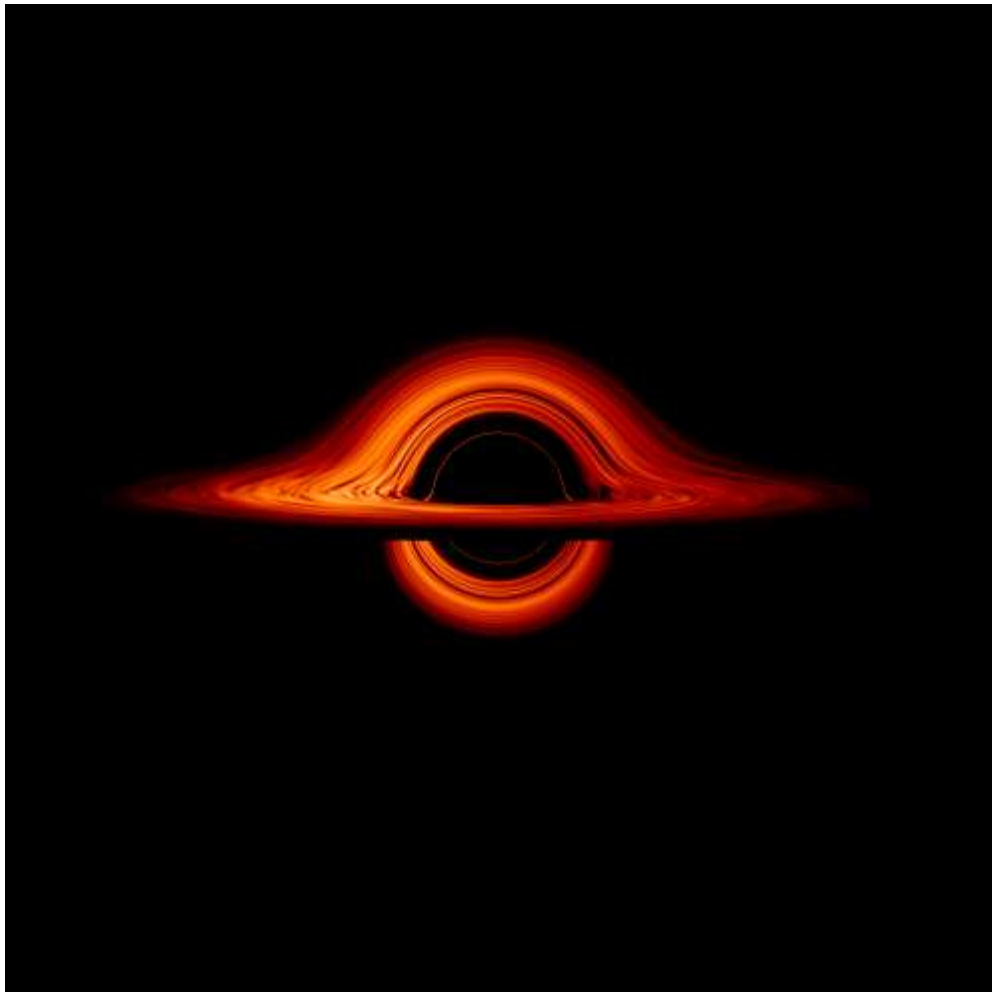
Lena – Originalfoto

Performanzanalyse – Pinktrees und Black white

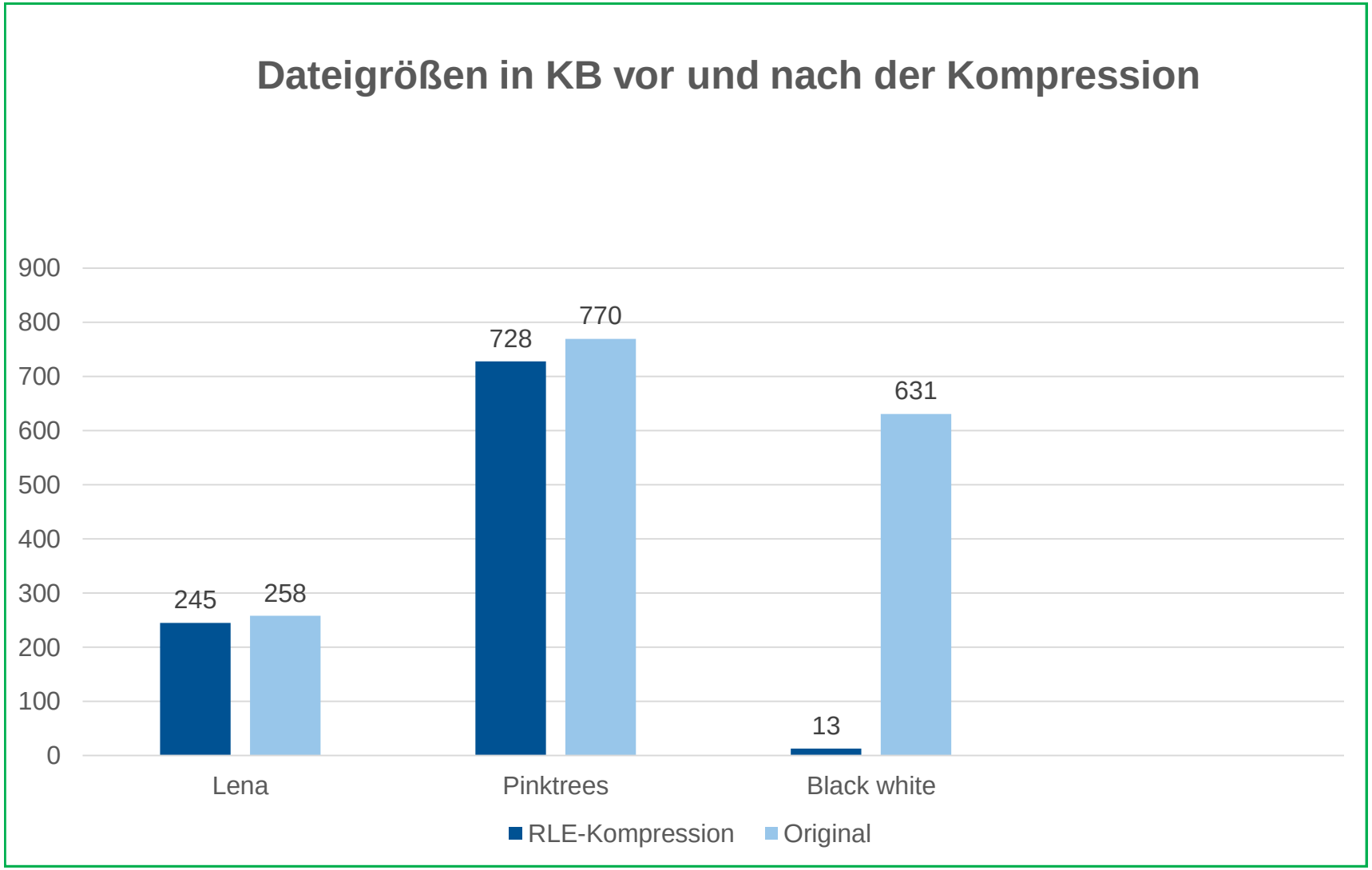




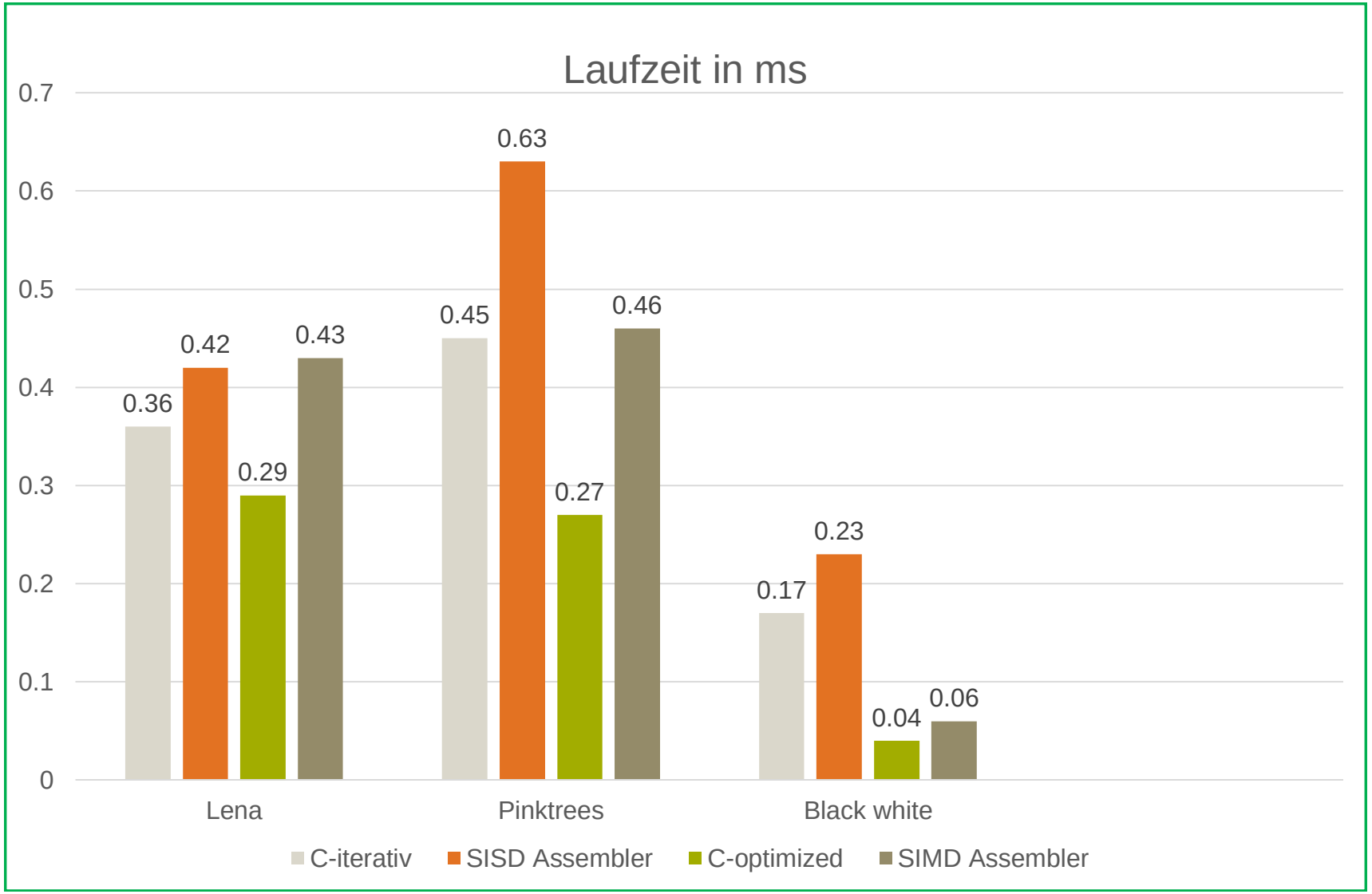
Performanzanalyse – Black hole



Performanzanalyse – getestete Dateigrößen



Performanzanalyse - Laufzeitvergleich



Performanzanalyse - Übersicht

- **Einfluss von Dateigrößen**
- **Kompressionsrate**
- **Naiver C Ansatz vs naiver Assembler Ansatz –** Compiler-Optimierungsstuffe 2
- **C mit memset und memcpy vs Assembler mit SIMD –** schneller als memcpy und memset fast unmöglich
- **Disassemblierung von memset und memcpy –** SIMD, ausgerichtete Speicherzugriffe, Pre-Fetching, Loop unrolling, weitere CPU-spezifische optimierungen

Zusammenfassung

- **Was wir gelernt haben …**

- **Ausblick – weitere Optimierungsideen** – in Assembly SIMD-Implementation auch weniger als 16 byte behandeln, weitere Optimierungmöglichkeiten für Schwarzweiß- und Zweifarbenbilder, Alignment des Speichers und Verwendung von moveAps anstelle von moveUps

- **Zusammenfassung –** Verständnis von BMP-Dateien, RLE-Verfahren, Kenntnisse über Zielarchitektur, GCC-Compiler, memset, memcpy und SIMD

Vielen Dank für Ihre Aufmerksamkeit!